ENGINEERING AND EVALUATING BULK DATA TRANSFER PLANNING OVER WIDE AREA NETWORKS

BY

SIMON P KRUEGER

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Adviser:

Associate Professor Indranil Gupta

Abstract

Users of cloud computing services are faced with the problem of planning and transferring large amounts of data across a geographically dispersed network under time and budget constraints. Pandora is the first service to solve the planning problem but underneath it has a set of issues, which are addressed in this work. This thesis enhances Pandora by improving its interface, scalability, modularity, and extensibility. This is done by evaluating the performance of various GPU linear program solvers on Pandora's workloads, and by designing and implementing two new systems: a modular C++ framework called Pandora's Toolbox that solves the bulk data transfer problem, and a web service for planning bulk data transfers. Pandora's Toolbox models the steps of the bulk data transfer problem and has an extensible design that allows various linear program solvers (e.g., GLPK and GPU solvers) to integrate into it. The web service effectively allows users to construct a shipping network and plan bulk data transfers over it through both a web and a RESTful interface.

To my Grandfather

Acknowledgments

I am grateful towards my family, my friends, and the Computer Science faculty and staff at the University of Illinois at Urbana-Champaign that have supported me over the past five years for helping me grow both intellectuality and personally. In particular, I am grateful towards Associate Professor Robin Kravets for giving me my first opportunity to do research, my advisor, Associate Professor Indranil Gupta, for all the guidance, support, and advice he has given me, and Brain Cho, for explaining the bulk data transfer problem to me and for mentoring me on the various aspects of being a graduate student. I thank Alex Loeb for reviewing this thesis and for providing moral support during the time of its writing.

Table of Contents

Chapter 1: Introduction

Users of cloud computing services are faced with the problem of transferring large amounts of data across a geographically dispersed network under time and budget constraints. They are faced with this problem because cloud computing services operate on the scale of terabytes and are composed of sites that are geographically dispersed and distant. There are many reasons to transfer data between sites. For example, users may have to perform initial transfers, cloud service provider migrations, site backups [49], or user data migrations. Furthermore, they want to perform their data transfers within a certain time deadline and they want to minimize the cost of the transfer.

Pandora [9] is the first service to solve this problem by first formulating a time expanded shipping network and then solving the network as a Mixed Integer Program (MIP). The time expanded shipping network forms a graph that is composed of sites, links, and a time deadline. This time expanded shipping network is then formulated into a MIP by placing its sites' and links' constraints into the MIP's matrix and vectors. The problem is NP-hard [9] and even a simple transfer problem with a small shipping network (i.e., small number of sites and links) will have a matrix with thousands of both rows and columns. A large matrix increases the computation time and the computation resources (i.e., memory) needed to solve the problem. Additionally, a problem with a large shipping network is difficult for

a human user to manage due to many site and link parameters. As a result, Pandora does not scale well when faced with a large shipping network and does not provide an accessible interface to manage the problem's parameters.

This thesis builds on top of the existing Pandora system by using GPU linear program solvers and by building a new web interface for it.

This thesis makes three major contributions: a web service to interface into the Pandora system, a new modular framework for the Pandora system, and evaluates the performance of GPU based linear program solvers on Pandora's workloads.

1.1 Pandora's web service

This thesis develops a web service that allows users to plan and view bulk data transfers online. There are several steps in planning an efficient bulk data transfer over a wide area network. First a user formulates their bulk data transfer by constructing a shipping network and specifying in that shipping network when and where the data is to be transfered. Then the formulated constraints need to be submitted to and processed by the Pandora system. Finally, once the Pandora system has processed the constraints of the bulk data transfer, the user needs to interpret the output into a shipping plan, which specifies the steps of the bulk data transfer that need to be performed in order to satisfy time and budget constraints. To effectively and easily allow users to formulate, submit, and view the results of their data transfer, this work has designed and implemented a web service. The web service provides a web site that is a human usable interface to plan data transfers, and a Representational State Transfer [18] (REST) endpoint that allows for a machine usable interface to plan data transfers.

1.2 Pandora's Toolbox

This thesis evaluates the performance of various linear program solvers in solving the bulk data transfer problem. In order to do this, the constraints of bulk data transfer problem must be placed into the linear program solvers via their Application Programming Interface (API) or via their supported file formats. However, most linear program solvers have different APIs and supported file formats. This variety of APIs and file formats makes it difficult to integrate new linear program solvers into the original Pandora system that is designed to use only one particular linear program solver. Furthermore, the original Pandora system has the components that formulates the problem's shipping network tightly coupled with the components that solve the shipping network as a linear program.

To address these issues, this thesis designs and implements Pandora's Toolbox, a C++ framework that is designed to be both extensible and modular. Pandora's Toolbox is advantageous because of its design considerations. Its extensiblility provides a way for various linear program solvers to plug in to the system. Additionally, it allows shipping networks to be efficiently managed in memory and makes the system more maintainable because it modularity separates the steps of the bulk data transfer problem in to independent components.

1.3 GPU LP Solvers

This work explores the benefits of using graphic processor unit (GPU) based linear program solvers by evaluating their running time in solving Pandora's workloads. In order to improve the computation time of solving a linear program (LP) with a large number of constraints, researchers have implemented parallelized LP solvers on GPUs [47, 3]. Indeed, evaluations of these LP solvers show that computation time is improved. However, the eval-

uations were only performed on randomized LP problems and have not been evaluated with LP problems from real applications. This thesis evaluates the performance of these GPU solvers on real applications by having them solve the bulk data transfer problem.

The bulk data transfer problem is converted into a large matrix that must be stored in a graphic card's memory. In order for a GPU solver to work on this large matrix, the GPU needs to have several gigabytes of memory. For this reason, the evaluations of the GPU solvers in this thesis are performed on NCSA's Lincoln GPU cluster [37]. This cluster has graphics cards with several gigabytes of memory and helps test the limits of the GPU solvers.

1.4 Thesis Outline

The rest of this thesis is outlined as follows: Chapters 2 and 3 look at the design, architecture, and implementation of the Pandora's web service and Pandora's Toolbox, respectively. After that Chapter 4 evaluates the performance of various LP solvers on Pandora's workloads. Then Chapter 5 discusses the similarities and differences of related work. Finally, Chapter 6 concludes the results of the thesis and briefly mentions future work.

Chapter 2: Web Service

This chapter discusses the design and implementation of a web service for planning bulk data transfers. The service provides both a human usable interface and a machine usable interface for planning transfers via a web site and a REST endpoint, respectively. The bulk data transfer over a wide area network problem determines the optimal way to transfer large quantities of data (i.e., $\geq 1\,\text{TB}$) over a Wide Area Network (WAN) under a certain time limit while minimizing the total cost of transfer. The WAN consists of a set of physical sites that are connected by internet links and shipping providers.

2.1 Pandora

People and Networks Moving Data Around [9] (Pandora) is the first system that solves the bulk data transfer problem[1]. Pandora takes in a shipping network, a deadline, set of source sites, and a destination site and computes the bulk data transfer problem. Pandora returns a total cost of transfer and a shipping plan that is composed out of a list of tuples that contain the amount of data to transfer, where data should be transfered from and to, the method of transfer, and time the transfer should take place.

[1]Pandora is a work done by Brian Cho, a University of Illinois at Urbana-Champaign PhD student in the Distributed Protocols Research Group (DPRG) advised by Associate Professor Indranil Gupta. Pandora, in its original incarnation, is not the author's work

Previously, the only way to interface with Pandora was through a command line program. This means that in order to use Pandora, users would have to be familiar with using a text based shell (aka Terminal). The command line version of Pandora had the user specify the number of sites, deadline time, sources, destination, and data demand as additional command line arguments. This makes for a high level of entry since some users may not have experience with using text based shells. Additionally, the old Pandora system would read in a graph file that described the shipping network. The graph file is space delimited and specifies the sites, links, and various parameters of the shipping network. The number of lines in a graph file was determined by the number of links in the shipping network so the graph files would quickly grow and become unmaintainable by users. Users would often have to write a script to generate and manage their graph files. Furthermore, there was no easy way to collaborate multiple shipping networks since user's graph files that specified the shipping network were completely separate. In order to collaborate users would have to merge their own graph files. Finally, the output of the command line system was overly verbose, difficult to understand, and hard to read. The output would contain internal system information about solving the problem that is unimportant to the user. Finally, a solved problem with a large number of links would create a long list of output that would be difficult to understand and visualize.

The Pandora web service is a way to alleviate the issues of the command line program. It provides a way for users of Pandora to easily use the service without having to know how to use a text based shell or install software. Users interact with the service by visiting Pandora's website or by using Pandora's REST endpoint. A high level overview of the architecture of the web service is depicted in Figure 2.1.

The website provides a human usable interface to the Pandora service. It allows users to

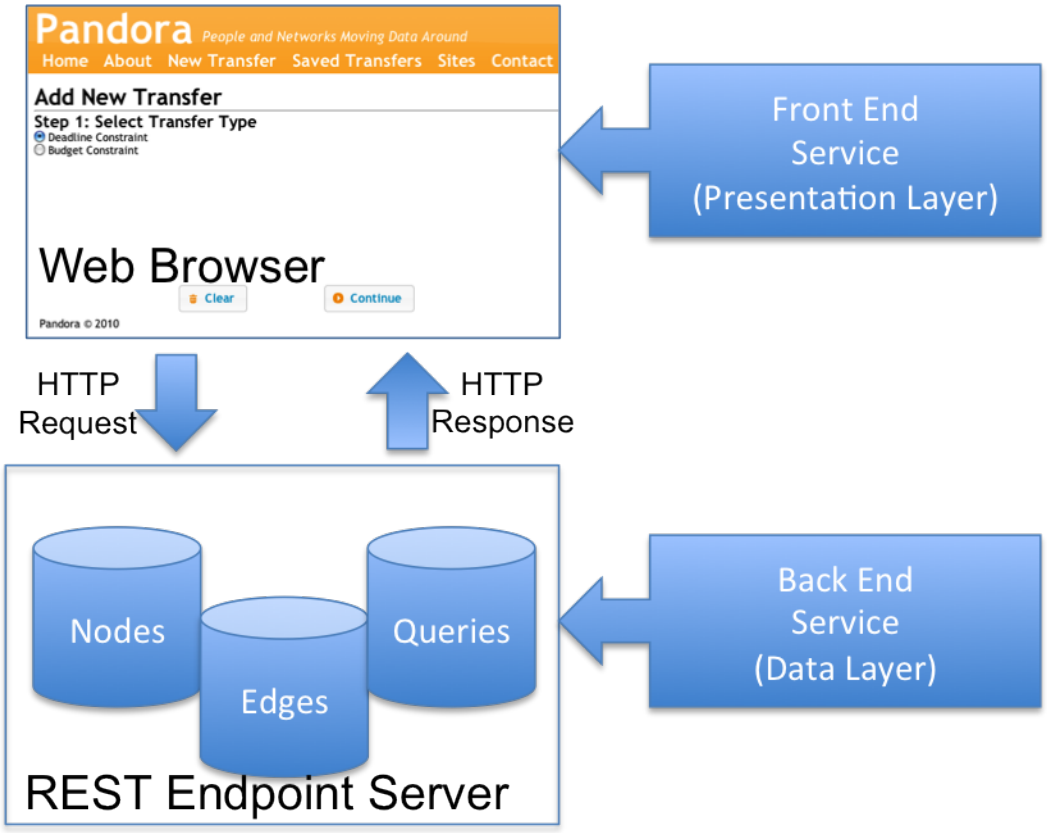


Figure 2.1: Overview of the Web Service

share a global shipping network and provides a form that allows them to enter the necessary information for their transfer. They first choose if they want the transfer to be a budget constraint problem or a deadline constraint problem. The budget constraint problem has the user specify the amount of money they would like to spend for the transfer. The deadline constraint problem has the user specify the start date and time, and end date and time. Then both types of problems have the user input each source site paired with the amount of data that is to be sent from the site, and the destination site. When the user submits her problem's constraints, the web service saves them as a query. This allows the user to easily view previous queries and prevent the need for users to resubmit queries. After the user's query is submitted and displayed, the web service's backend begins processing the query. Once the job is finished, the user can view their results, which contains the cost of

the transfer, the computation time, and a shipping plan. The shipping plan is a list of links used that detail the source site and destination of the link, the data's departure and arrival time, and the amount of data to send on the link.

2.2 REST Endpoint

The core of the web service follows a Representational State Transfer [18] (REST) style of architecture that provides a machine usable interface to the Pandora service. It is built on top of Hyper Text Transfer Protocol [17] (HTTP), which allows any programing environment that supports HTTP to interface to the service. This decouples Pandora's model from its presentation, which offers great flexibility for development and users. It provides greater flexibility because it allows users to interact with the web service in a way that they prefer. For example, users could write their own programs that communicate with the Pandora web service. They could automate their access rather than relying on a human usable interface that the web site provides. Additionally, users could write a new front end application to support their own needs, preferences, and/or platform (e.g., Android [31] or iOS [27]).

The endpoint constructs a virtual shipping network composed of physical sites and links that users use to construct a shipping plan on. The virtual shipping network, and shipping plans are represented in the web service by three main entities: Nodes, Edges, and Queries. All three entities provide ways to interface to the service and are listed in the root of the endpoint.

2.2.1 User

A subtle yet important distinction to make is the difference between the human user of the Pandora service and the machine user of the REST endpoint. For this section the term

user, in regards to the users of the REST endpoint, is defined as the agent(s) that make HTTP requests on behalf of the human user interested in planning a bulk data transfer. For example, the canonical user of the REST endpoint for this work is a web browser via its XMLHttpRequest (XHR) Application Programming Interface (API). A human user that is interested in planning a bulk data transfer uses the web service's frontend through a web browser. The web browser acts on behalf of the human user and is the user of the web service's REST endpoint. The human user decides which actions and operations to perform and delegates them to the web browser. The web browser then performs the actions and operations through the REST endpoint by interfacing to it through HTTP. An alternate example of a user of the REST endpoint could be a program written in Java that acts on behalf of a human user.

2.2.2 Data Format

All of the data sent through the REST endpoint is serialized and deserialized using JavaScript Object Notation [12] (JSON) since it is minimalistic, well supported, and elegantly integrates with JavaScript. Choosing JSON increased development productivity because the core portion of the frontend was written in JavaScript and JSON objects are treated as JavaScript objects and vice versa. An alternative format that could be included in future versions of the system is the eXtensible Markup Language [4] (XML) format. XML was not chosen since it was more verbose and complex than JSON. Also, it does not integrate into JavaScript as nicely as JSON. However, alternative data formats do provide greater development flexibility since developers using the endpoint can choose their preferred format.

2.2.3 Service Document

The root of the web service is called the service document because it lists all the available

services that are provided by the system and their Uniform Resource Locators [2] (URLs). Users will first need to perform a HTTP GET on the service document to retrieve the URLs of the other entities. An example of the service document is shown in listing 2.1.

Listing 2.1: JSON payload response from a HTTP GET at root of the end point

```
{
  "nodes": "http://hillary.cs.uiuc.edu/api/nodes/",
  "edges": "http://hillary.cs.uiuc.edu/api/edges/",
  "queries": "http://hillary.cs.uiuc.edu/api/users/1/queries/"
}
```

2.2.4 Nodes

Nodes represent Sites in a shipping network. A Node has an identification number, name, street address, and self link. The identification number is to uniquely identify the node, the name is an arbitrary string but often will be the domain name of the site it represents, the address is used to calculate shipping costs, and the self link is used to perform operations on to the node. Users can retrieve the set of Nodes by performing a HTTP GET on the node's URL that is specified in service document. An example response from a HTTP GET on the Nodes URL is shown in listing 2.2. In order to add and remove Nodes, administrators can perform HTTP POST and HTTP DELETE on the node's URL. Adding and removing nodes is restricted to administrators because nodes are global and shared by all users. If an arbitrary user was able to add and remove nodes this could adversity affect the other users of the system because a removed node could still be in use by another user's shipping plan.

Listing 2.2: Example JSON payload response from a HTTP GET on the nodes end point

```
{
  "nodes": [{
      "city": "Palo Alto",
      "state": "CA",
      "street": "241 Panama Street pine Hall , Room 115",
      "updated": "2010-12-27T14:55:37",
      "name": ".scs.stanford.edu",
      "countryCode": "US",
      "created": "2010-12-27T14:55:25",
      "id": 2,
      "selfLink": "http://hillary.cs.uiuc.edu/api/nodes/2/",
      "zipCode": "94305"
    }, {
      "city": "Urbana",
      "state": "IL",
```

```
      "street": "1304 West Springfield Avenue",
      "updated": "2010-12-27T14:55:37",
      "name": ".cs.uiuc.edu",
      "countryCode": "US",
      "created": "2010-12-27T14:55:25",
      "id": 18,
      "selfLink": "http://hillary.cs.uiuc.edu/api/nodes/18/",
      "zipCode": "61801"
    }
  ]
}
```


2.2.5 Edges

Edges represent Links inside of a shipping network. Edges contain an identification number, source site, destination site, cost, capacity, transit time, link type, and a self link. The identification number is to uniquely identify the edge. The source site is the location the data is coming from and the destination site is where the data is going to. The cost is the amount in USD that it takes to use the link. The capacity is the amount of MBs that can be sent over the link in one time unit. The transit time is the number of hours it takes to go from the source to the destination. The link type specifies whether the edge is a shipping link or an internet link. The self link is a URL that allows operations to be done on to the edge through HTTP GET, POST, and DELETE methods. An example of the response to a HTTP GET on the edges endpoint is shown in listing 2.3. Similar to Nodes, Edges are shared amongst all the users of the system. Users can retrieve the list of edges by performing a HTTP GET on the edges endpoint. Only an administrator may add and remove edges by performing a HTTP POST and HTTP DELETE on the edges endpoint.

Listing 2.3: Example JSON payload response from a HTTP GET on the edges end point


```
{
  "edges": [{
      "updated": "2010-12-27T14:58:00",
      "cost": 8.81,
      "transferMethod": "gro",
      "transitTime": 96,
      "functionType": "step",
      "id": 41,
      "capacity": 2147483647,
      "created": "2010-12-27T14:55:25",
      "destination": {
        "city": "Urbana",
        "state": "IL",
        "street": "1304 West Springfield Avenue",
        "updated": "2010-12-27T14:55:37",
```

```
        "name": ".cs.uiuc.edu_i",
        "countryCode": "US",
        "created": "2010-12-27T14:55:25",
        "id": 12,
        "selfLink": "http://hillary.cs.uiuc.edu/api/nodes/12/",
        "zipCode": "61801"
      },
      "source": {
        "city": "Palo_Alto"
        "state": "CA"
        "street": "241_Panama_Street_pine_Hall,_Room_115"
        "updated": "2010-12-27T14:55:37"
        "name": ".scs.stanford.edu"
        "countryCode": "US"
        "created": "2010-12-27T14:55:25"
        "id": "13"
        "selfLink": "http://hillary.cs.uiuc.edu/api/nodes/13/"
        "zipCode": "94305"
      },
      "selfLink": "http://hillary.cs.uiuc.edu/api/edges/41/"
    }
  ]
}
```


2.2.6 Queries

A Query represents a bulk transfer request and contains all the information needed to construct a transfer plan: a list of source sites paired with the amount of data to be sent from the corresponding site, the destination site, the requested arrival date and time, and the requested departure date and time. Additionally, queries contain an identification number, a results link which is the location of the query's result, and a self link which provides a way to perform operations on to a query. An example of the response given to a HTTP GET on the queries end point is shown in listing 2.4. Users can add their own queries by performing a HTTP POST on the queries endpoint. In order to modify a query, users can perform a HTTP POST on the query's self link and to remove a query users can perform a HTTP DELETE on the query's self link. Each user has their own independent set of queries and are not allowed to view, modify, or delete other user's queries. This is to preserve privacy amongst all the users. If a user could view any other users' shipping plans, that user could infer about other users' possibly private operations.

Listing 2.4: Example JSON payload response from a HTTP GET on the queries end point


```
{
  "queries": [{
    "updated": "2011-04-10T14:40:46",
```

```
    "delta": 1,
    "sources": [{
      "node": {
        "city": "Durham",
        "state": "NC",
        "street": "334 Blackwell St. Suite 2106",
        "updated": "2010-12-27T14:55:37",
        "name": ".cs.duke.edu",
        "countryCode": "US",
        "created": "2010-12-27T14:55:25",
        "id": 1,
        "selfLink": "http://hillary.cs.uiuc.edu/api/nodes/1/",
        "zipCode": "27701"
      },
      "demand": 1000000
    }, {
      "node": {
        "city": "Albuquerque",
        "state": "NM",
        "street": "2701 Campus Blvd. NE",
        "updated": "2010-12-27T14:55:37",
        "name": ".unm.edu",
        "countryCode": "US",
        "created": "2010-12-27T14:55:25",
        "id": 9,
        "selfLink": "http://hillary.cs.uiuc.edu/api/nodes/9/",
        "zipCode": "87131"
      },
      "demand": 1000000
    }],
    "user": 1,
    "arrivalTime": "2011-04-14T17:00:00",
    "id": 9,
    "departureTime": "2011-04-11T17:00:00",
    "created": "2011-04-10T14:40:46",
    "destination": {
      "node": {
        "city": "Urbana",
        "state": "IL",
        "street": "1304 West Springfield Avenue",
        "updated": "2010-12-27T14:55:37",
        "name": ".cs.uiuc.edu",
        "countryCode": "US",
        "created": "2010-12-27T14:55:25",
        "id": 18,
        "selfLink": "http://hillary.cs.uiuc.edu/api/nodes/18/",
        "zipCode": "61801"
      },
      "demand": 2000000
    },
    "resultsLink": "http://hillary.cs.uiuc.edu/api/users/1/queries/9/results/",
    "selfLink": "http://hillary.cs.uiuc.edu/api/users/1/queries/9/"
  }]
}
```


Query Results

A Query Result represents the status of a query operation and the shipping plan to be performed for a solved query's transfer request. A query's result contains the time in seconds the service took to solve the query, the cost in USD to perform the transfer, and a list of edges paired with additional information. Each edge pair contains the arrival date

and time, the departure date and time, and the amount of megabytes to transfer on the edge. Additionally, the query's result contains a progress variable and a solvable variable. The progress variable specifies the percent of completion on the query, where 0% means the query has not been started and 100% means the query has been completely processed. The solvable variable indicates if the given query is solvable. Certain queries may not be solvable because the query may be trying to send a large amount of data over a short period of time that is not feasible for the given shipping network. Query results are obtained by performing a HTTP GET on the query's resultLink. An example of a query result is displayed in listing 2.5.

Listing 2.5: Example JSON payload response from a HTTP GET on a query's resultLink

```
{
  "updated": "2011-04-10T14:40:53",
  "solvedTime": 0.338364,
  "created": "2011-04-10T14:40:46",
  "edges": [{
    "arrivalTime": "2011-04-11T19:00:00",
    "departureTime": "2011-04-11T19:00:00",
    "destination": { ... },
    "source": { ... },
    "flow": 28966,
    "linkType": "internet"
  }],
  "cost": 0.01,
  "progress": 100,
  "solvable": true
}
```

2.3 Frontend

The frontend of the service is written in eXtensible HyperText Markup Language [1] (XHTML), Cascading Style Sheets [40] (CSS), and JavaScript [35]. All of these technologies are used to compose a website that provides a graphic user interface to the REST endpoint. Specifically, a user makes a request to the frontend HTTP server and it responds with the corresponding XHTML, CSS, and JavaScript content. Then the content that is fetched from the HTTP server is loaded in to the user's web browser and the user begins their session. In order to make the service more interactive and responsive the JavaScript code makes asynchronous HTTP calls (i.e., Asynchronous JavaScript and XML (AJAX)) to the backend

REST endpoint. Based on the response the JavaScript code receives, it may manipulate the web site's Document Object Model (DOM). Additionally, to make the web site aesthetically appealing jQuery UI [44] is used because it provides animations, icons, and widgets.

Users of the frontend web site first formulate a transfer plan that specifies all the parameters and constraints of a particular bulk data transfer. The user interface of the web site is shown in Figure 2.2 and Figure 2.3. The process is broken down in to a wizard like system where each point to enter parameters and constraints is broken in to a separate step. The first step is to determine the type of transfer to perform (See Figure 2.2a). This can either be a deadline constrained problem where the transfer needs to be completed by a certain time deadline while optimizing the cost, or a budget constrained problem where the transfer needs to cost less than or equal to a particular budget while optimizing the time to complete the transfer. The second step depends on the choice chosen in the first step. If a deadline constraint type was selected then the user must enter a start date and time and an end date and time (See Figure 2.2b). Else if a budget constraint type was selected then the user must enter the budget amount in USD (See Figure 2.2c). The third steps chooses the destination site (See Figure 2.2d). Since it is obtrusive to list a large number of sites for the user to choose, an auto-complete box is placed on the form so the user can type the site they wish to use and all matching sites will be listed. Finally, the forth step is to enter all the source sites paired with the amount of data to send from them (See Figure 2.2e). Similar to the third step, an auto-complete box is listed to help select sites. Since the number of source sites is arbitrary and dependent on the user's constraints, an "Add Another Source" button is listed to allow the user to add an arbitrary number of source sites. A "Remove" button is placed underneath each source site that allows users to remove sites that were mistakenly added. A user must always list at least one source site and demand pair, so a "Remove" button is not placed underneath the first source site input box. Forward and Back buttons are added to the wizard to allow the user to move between steps before submitting the transfer plan. Once the user is confident about their input they can submit the transfer to plan to

be processed. Upon success, the web page displays a successful submit notification and the submitted transfer plan is listed on a separate page that lists all the transfers the user has ever submitted (See Figure 2.3a). To view the results of the recently submitted transfer plan the user can go to this page, select their transfer, and click view results (See Figure 2.3b). If the transfer is solvable and has been successfully processed a transfer plan and cost of transfer is listed. Additionally, from this page the user can delete transfer plans.

2.4 Backend

The backend is responsible for receiving, processing, and managing user requests. A depiction of the backend's workflow is shown in Figure 2.4. When a request is sent to the backend's HTTP server, the URL of the request is placed through multiple regular expressions to determine the python function that shall handle the request. If the request's URL does not match any of the regular expressions a HTTP 404[2] response is returned. Else if the request's URL matches a regular expression the request is put through a function who's behavior is determined by the request's type (e.g., HTTP GET, HTTP POST, HTTP DELETE). If the request is a HTTP GET, most functions will retrieve an entry from the MySQL database and serialize the entry into JSON and send it in a HTTP 200[3] response. If the request is a HTTP POST, most functions will either create or modify entries in the MySQL database and respond with a HTTP 200 or HTTP 201[4] response on success. If the request is a HTTP DELETE, most functions will remove entries in the MySQL database and respond with a HTTP 204[5]. If any of the functions encounter an error while processing a request (e.g., the requested entry does not exist, the request was improperly formated), the functions will return the proper HTTP response code that corresponds to the encountered error.

When a new query is created (via a HTTP POST), a new job is placed on a synchronized

[2]Not Found Error Message
[3]Standard response for successful HTTP requests
[4]Response for when a request has been fulfilled and resulted in a new resource being created
[5]Response for a successfully processed request, but no content is being returned

queue. Worker threads periodically take jobs off this queue and block when the queue is empty. A worker thread is responsible for processing jobs by building a graph file that represents the shipping network based on the user's constraints and giving the graph file and the job's constraints to the Pandora Solver Engine (See Chapter 3). The worker thread waits for the engine to solve the problem and return the results. Once the engine is finished, the worker thread parses and stores the results to persistent storage.

The backend is written in Python using the Django [23] web framework and runs on top of an Apache HTTP Server [22] with mod_wsgi [14]. All of the data in the backend is stored in a MySQL [11] database. Python and Django were chosen because they both allow for rapid development of web services. Additionally, Django interfaces to multiple persistent data stores, so changing persistent stores for development reasons or for production reasons becomes trivial. For example, if the service needed to increase scalability, the persistent store could be changed from MySQL to an enterprise Orcale database, or even better, a distributed data store like Cassandra [38, 46], BigTable [7, 33] through Google App Engine [32], or CouchDB [20, 8].

2.5 Offline Processing

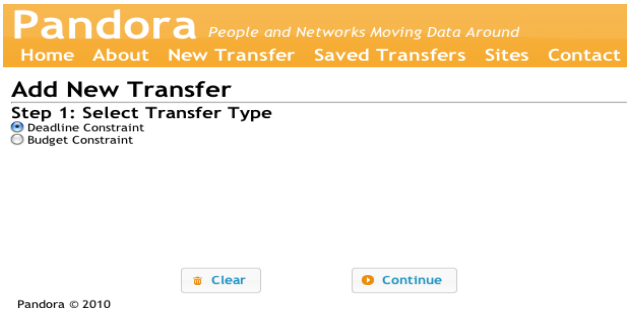
Part of the work needed to run the web service is processed offline. Specifically, the web service needs the cost of shipping disks to various locations. The cost of shipping is determined by the shipping provider, the weight of the shipment, the distance traveled, and the type of service used. The web service uses FedEx's [10] and UPS's [34] Simple Object Access Protocol (SOAP) API to gather shipping costs. The web service weekly updates all of the cost of shipping links by querying the shipping providers' SOAP service and saving their responses in the backend's persistent storage. Disks often weigh about 6 pounds (LBS) and are shipped using the shipping providers' ground service, overnight service, and second day service. Depending on the location, overnight service shipments usually cost between 20 United States dollars (USDs) and 70 USDs, two day service shipments usually cost between

10 USDs and 30 USDs, and ground service shipments usually cost between 5 USDs and 10 USDs.

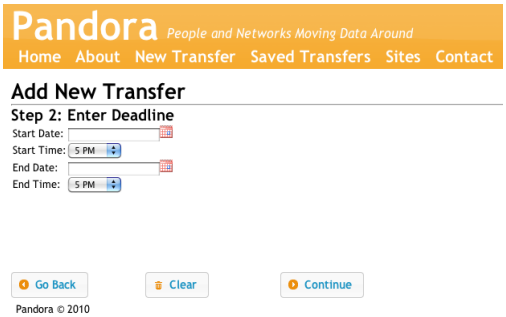
2.6 User Account Management

Each user of the system need to have their own separate account because each user will have their own set of private queries. In order to do this, the web service needs an user account management system. One option is to use Django's user management module. However, it is a security risk for the web service to house user's account information (e.g., passwords, personal information). For example, in 2010 a popular gadget and technology news web site, Gizmodo[6], had their users' passwords stolen [24]. In order to prevent a scenario that compromises the users' passwords and other personal information, the web service does not store user's account information. Instead the web services uses Facebook's Authentication [28] service to authenticate users. This way all of the user's personal information is stored on Facebook's machines and all the authentication and authorization is handled through the OAuth 2.0 protocol [15]. Once the user is authorized and authenticated through Facebook, a user account is created in the web service with a user name that is the same as the user's Facebook user id, and a password that is a hash of the web service's secret key and the user's Facebook user id. This way the only personal data of a user that is stored on the web service is the user's Facebook user id, which is not private information as the Facebook service publicly discloses their users' ids.

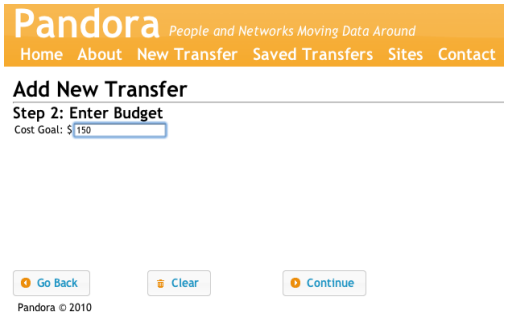
[6] `http://gizmodo.com/`



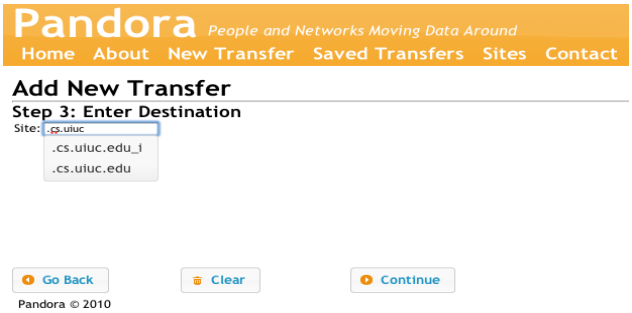
(a) Step 1: Select Transfer Type



(b) Step 2a: Enter Deadline



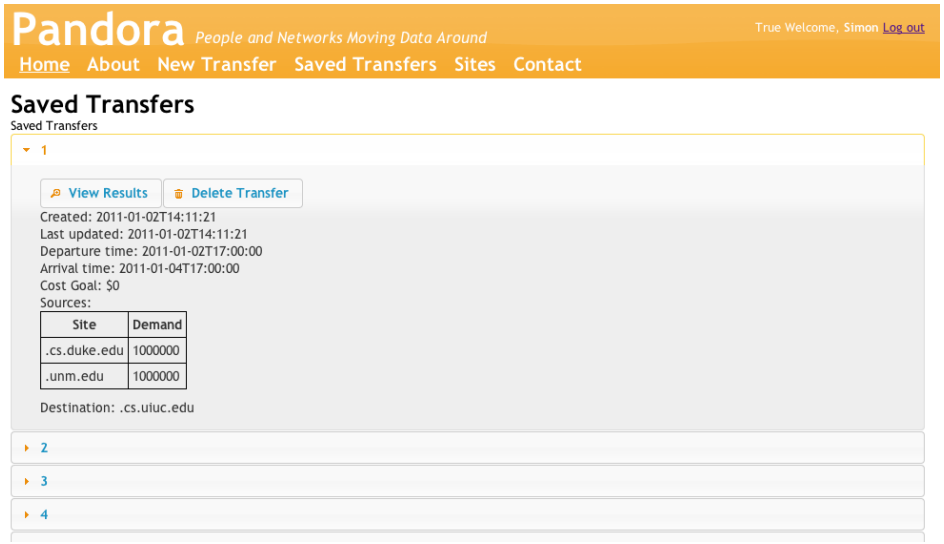
(c) Step 2b: Enter Budget



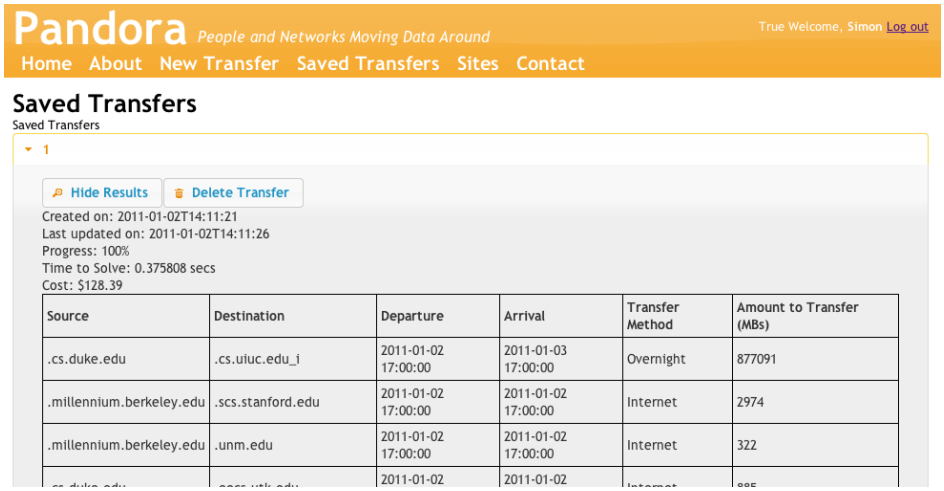
(d) Step 3: Enter Deadline

(e) Step 4: Enter Sources

Figure 2.2: Steps to plan a transfer



(a) List of saved transfer plans



(b) The cost and shipping plan for a processed transfer

Figure 2.3: Saved Transfers Results

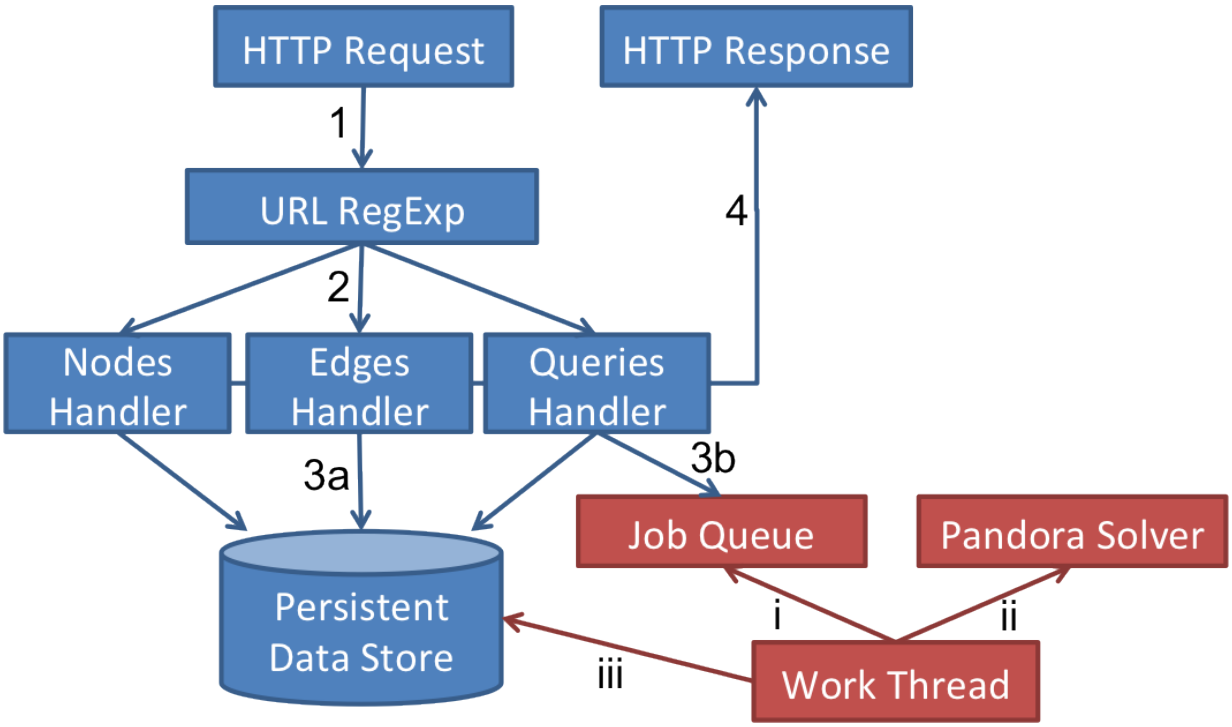


Figure 2.4: Backend Workflow. Step 1: The HTTP Request is sent to the RegExp Handler. Step 2: The RegExp handler uses the URL to determine who should handle the request. Step 3a: The approperiate handler acts on the persistant data store. Step 3b: If a new query is being posted a new Job is put on the Job Queue. Step 4: After the handler is finished it sends back the proper HTTP Response. Step i: The work thread pulls jobs off the job queue. Step ii: The workthread sends the job over to the Pandora Solver. Step iii: The workthread stores the results to the Persistent Data Store.

Chapter 3: Pandora's Toolbox

In order to effectively use each linear program solver on the bulk data transfer problem, this thesis designs and implements Pandora's Toolbox, a modular and extensible C++ framework that solves the bulk data transfer problem. Specifically, Pandora's Toolbox is designed to decouple the shipping network formulation from the linear program solvers that solve the final step of the bulk data transfer problem. It also separates each step in solving the bulk data transfer problem in to independent components.

There are several advantages to Pandora's Toolbox: it is easier to maintain, it provides better performance, and it allows new linear program solvers to be easily integrate into the system. It is easier to maintain because its independent components are concentrated and focused on a specific task. Thus, it makes components easier to manage, reason about, and test. Performance is improved because independent components allow a shipping network to be operated on multiple times by maintaining it in memory. The old system would need to read a shipping network from disk into main memory for every single operation performed on a shipping network. Pandora's Toolbox is able to integrate various linear program solvers into the system through an extensible solver component. This allows various linear program solvers to easily plug in to the system by simply implementing the solver component's interface. Being able to easily extend the system in this way is very important to this thesis because Chapter 4 evaluates the performance of various linear program solvers.

The original Pandora system [9] did not intend to use more than one linear program solver. The design of the original Pandora system makes it difficult to integrate various

linear program solvers. Specifically, its design is monolithic with several components highly coupled together. The code in the original Pandora Solver did not cleanly separate the steps in solving the bulk data transfer problem either. The system intertwined the following steps into one component: 1) reading in sites and links from persistent storage, 2) constructing the shipping network, 3) converting the shipping network in to a time expanded shipping network, and 4) solving the time expanded shipping network via a linear program. This made it difficult to maintain the system and most importantly, this made it difficult to change linear program solvers. The original system would place the whole shipping network immediately into GNU Linear Programing Kit's [29] (GLPK) Application Programming Interface (API). Once this happens, it becomes difficult to modify the shipping network because the level of abstraction changes from a high-level abstraction where modifications are performed on sites and links to a low-level abstraction where modifications are performed on matrices and vectors. Often there are cases where modifying the shipping network in its intermediate state is beneficial. For example, rather than reading from disk and constructing the same shipping network for each query, the shipping network can be read from disk and constructed once. Then the shipping network can be maintained in a machine's memory and queried upon with different source, destination, and time constraints. Additionally, if a shipping network needs to be modified (e.g., add, remove, or modify a site or link), rather than modifying the persistent data store and reloading the whole shipping network, the modifications can be performed within memory in an incremental manner.

Pandora's Toolbox fixes the coupling problem of the original Pandora Solver by decoupling each step in to its own set of C++ interfaces and classes. These interfaces and classes model the problem at a high-level and provide extensibility. Pandora's Toolbox consists of four main components: 1) a Shipping Network Builder module, 2) a Shipping Network module, 3) a Time Expanded Shipping network module, and 4) an extensible Solver module. The design of Pandora's Toolbox is depicted in Figure 3.1. The Shipping Network Builder component reads in the sites and links from a persistent data store (see Chapter 2) and uses

them to build a Shipping Network. The Shipping Network component represents the physical shipping network that Pandora uses to plan data transfers on and is composed of Sites, Shipping Links, and Internet Links. The Time Expanded Shipping Network component represents the expanded form of the Shipping Network where Sites and Links are expanded in to a more abstract form of Nodes and Edges based on the problem's constraints. The Solver component takes in a Time Expanded Shipping Network and places the Time Expanded Network Shipping Network in to a linear program solver's API or a file format that a linear program solver can use.

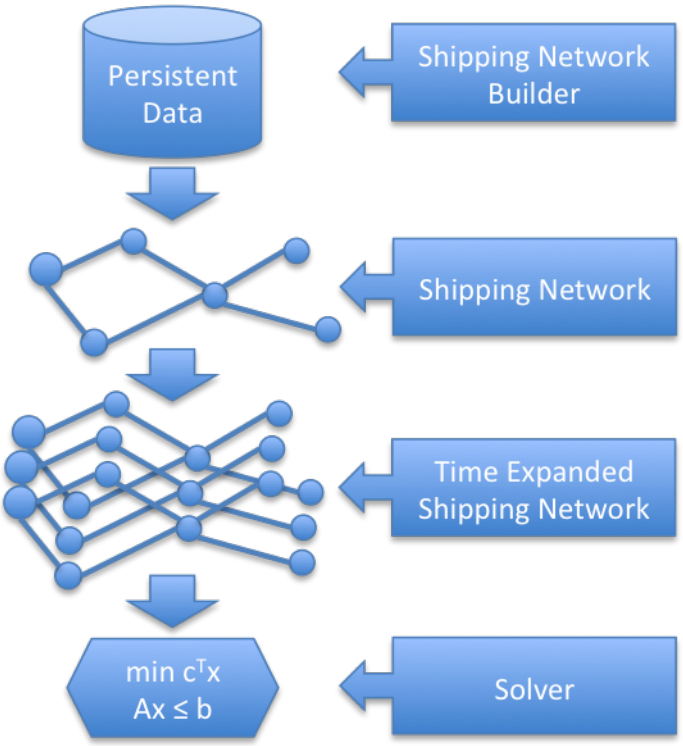


Figure 3.1: The Design of the C++ Framework

3.1 Shipping Network Builder

The job of the Shipping Network Builder Module is to read in Sites and Links from persistent storage and then use them to form a Shipping Network. The persistent storage could be a separate data file or it could be the data store used in the web service (See

Chapter 2). The point of the module is to automate the process of building the Shipping Network and have it separate the data sources of the system from the core structures of the framework.

3.2 Shipping Network

The Shipping Network is a representation of the physical shipping network of the system. It is composed of a set Sites and a set Links. A Unified Modeling Language (UML) class diagram of the classes used in the Shipping Network is shown in Figure 3.2.

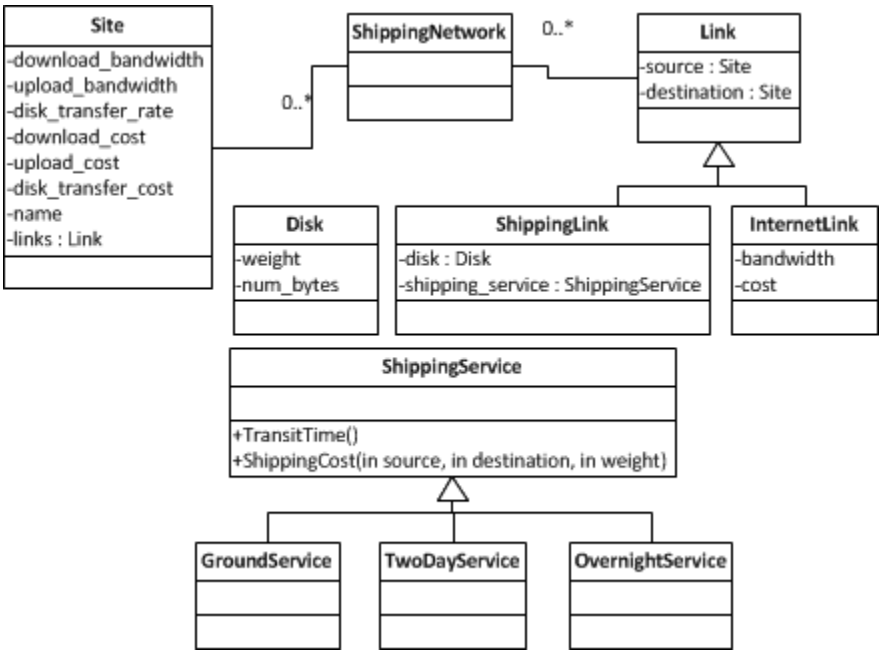


Figure 3.2: Shipping Network Class Diagram

A Site has a name, download bandwidth, upload bandwidth, disk transfer rate, download cost, upload cost, disk transfer cost, and a set of links. The download bandwidth and upload bandwidth represent how much data the Site can receive and send through its internet connection, respectively. The download cost and upload cost are the costs associated with receiving and sending data through the Site's internet connection, respectively. The disk transfer rate and disk transfer cost are used to model the transfer rate and transfer cost of transferring data from a shipped disk to the Site's machines, respectively. The set of links

are all of the site's incoming and outgoing links.

There are two types of Links in the Shipping Network: a Shipping Link and an Internet Link. Shipping Links contain a disk, and a shipping service. The disk represents a physical hard disk drive and it has a fixed number of bytes and weight. The disk is shipped through the shipping service which can be either ground, two day, or overnight. Internet Links have a bandwidth that limits the amount of data that can be sent over the link, and a cost. Both types of Links have a source Site and destination Site.

3.3 Time Expanded Shipping Network

The Time Expanded Shipping Network is a transformed version of a Shipping Network. It is composed of Time Expanded Nodes, Time Expanded Edges, a total time, a sink Site, and a list of source Sites paired with an amount of data to be transfered from them. Time Expanded Nodes are constructed for each Site and for each point in time. Time Expanded Edges connect Time Expanded Nodes across different points in time. The total time represents the total allocated time to perform the transfer in. For example, if the total time was 48 hours then the transfer must be completed in 48 hours or less. The sink Site is the Site where all the data is being transfered to and the source Sites are the Sites where the data is being transfered from. Additionally, during construction of the Time Expanded Shipping Network, Shipping Links in the Shipping Network are expanded into a particular arrangement of Time Expanded Nodes and Time Expanded Edges. A UML class diagram of the classes used in the Time Expanded Shipping Network is shown in Figure 3.3.

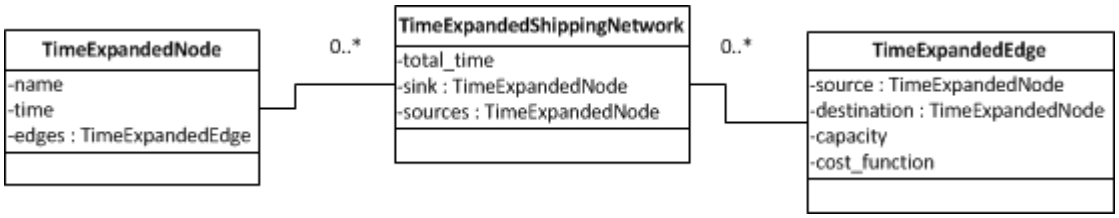


Figure 3.3: Time Expanded Shipping Network Class Diagram

Time Expanded Nodes represent a Site at a particular point in time. A Time Expanded Node has a name, time point, and set of edges. The name matches the Site's name and is used to identify nodes, and the time point represents the Site in a particular instant in time.

Time Expanded Edges connect Time Expanded Nodes, and contain a capacity and cost function. Additionally, since they connect two Time Expanded Nodes, the difference of the source's time and the destination's time represents the transit time of the Time Expanded Edge.

3.4 Solver

The Solver is the module that solves a Time Expanded Shipping Network. There is no concrete implement of a Solver because it is an interface (i.e., a C++ abstract base class with only pure virtual methods) that each separate linear program solver needs to implement. This is so a new linear program solver can be added to the system at any time by writing a new independent subclass of Solver that calls the new linear program solver's API. For this thesis a GLPK Solver, GPU Simplex Solver, and CUDA LP (culp) Solver were implemented. An UML class diagram of the classes used in the Solver is shown in Figure 3.4.

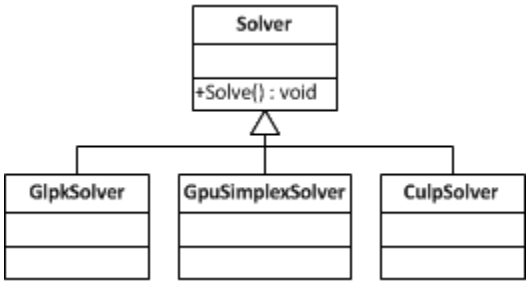


Figure 3.4: Solver Class Diagram

The GLPK Solver places the entire Time Expanded Shipping Network into GLPK's API by constructing the rows, columns, and their constraints. Both the GPU Simplex Solver and CUDA LP Solver place the entire Time Expanded Shipping Network into file formats that their corresponding linear program solvers can use. The process of placing the Time

Expanded Shipping Network into the various linear program solver's APIs and file formats involves placing the Time Expanded Shipping Network's Nodes and Edges into a modified version of the canonical linear programing form (See Chapter 4).

Chapter 4: Evaluation

This chapter evaluates the performance of Graphic Processing Unit (GPU) linear program solvers in solving the bulk data transfer problem. Linear programming problems consists of many linear equations that can be operated on independently. This makes GPUs great at solving them because they have hundreds of cores that can easily perform parallel computations. To prove this, researchers have implemented various linear programming solvers on a GPU [47, 3]. Indeed, evaluations of these solvers show improved performance.

The bulk data transfer problem is a real application that could benefit from the GPU solvers because the core of it is a linear programming problem. However, it is unclear how they would perform since the evaluations were only performed on randomized linear programming problems and not on linear programming problems from real applications. To further evaluate the performance of the GPU solvers, this chapter has them solve the bulk data transfer problem.

The evaluation is done by first forming sample shipping networks that transfer varying amounts of data with varying time deadlines and sources. Then each linear program solver is evaluated on two factors: the size of its representation for each shipping network in matrix form, and on the computation time to solve each of the shipping networks.

Our evaluations are performed on a Nvidia Tesla S1070 GPU with 16 GB of memory. This is necessary because shipping networks become large quickly and thus require large amounts of data to be placed into memory at one time. Finally, in order to operate on the bulk data transfer problem, each GPU solvers implements the Solver interface according to

its own API and/or supported file formats (see Chapter 3.4). This approach cleanly separates the code of each GPU solver and it cleanly separates the code that solves the linear program from everything else.

4.1 Linear Programming

In order to understand, the evaluation it is important to understand the basics of linear programing and how the bulk data transfer problem is formulated in to a linear program (as originally stated in [9]). The canonical form of a linear program tries to maximize $c^T x$ subject to $Ax \leq b$, where A is a $m \times n$ matrix. A bulk data transfer planning problem minimizes $c^T x$ and places its constraints into x, c, b, and A. $x_i \in x$ represents the amount of data to be shipped across link i. $c_i \in c$ corresponds to a cost function of sending x_i amount of data through link i. In A there are two types of rows: conservation rows and capacity rows. A conservation row in A represents a site's constraint of having incoming flow equal the outgoing flow, except for source and sink sites, which have unequal incoming and outgoing flows. A capacity row in A represents a link's constraint on the amount of data it can send. This makes A a vary sparse matrix with mosts rows having the number of non-zero coefficients much less than the number of columns in n. Depending on a solver's implementation, a sparse matrix can be beneficial, as sparse matrices can be stored in special formats to conserve space. Given this information, the number of columns in A and the size of the x vector are determined by the number of links in a time expanded shipping network. The number of rows in A and the size of the b vector are determined by the number of sites and the number of links in a time expanded shipping network.

4.2 Shipping Network

The evaluations were performed on a shipping network that models a real world shipping network with 10 different universities acting as sites. The Shipping Network used in the

evaluations is shown in Figure 4.1. This shipping network is a similar to the shipping network used in the Pandora paper [9]. The 10 universities that make up the shipping network are: University of Illinois at Urbana-Champaign, Standford University, University of California at Berkley, University of New Mexico, Duke University, University of Rochester, Washington University in St. Louis, University of Tennessee at Knoxville, Kansas State University, and University of Kansas. In the experiments the shipping network only ever has one sink site, which is University of Illinois at Urbana-Champaign. Conversely, the number of source sites vary for each experiment, ranging from 1 to 8 source sites. However, only sources and sinks are placed in the shipping network for the experiment. That is, if a site is not a sink or a source then it is removed from the shipping network. The data to be transfered is divided evenly across all the source sites. Finally, the shipping network forms a complete graph where every node (i.e., site) is connected to every other node via an edge (i.e., internet link) making a shipping network with n sites have $n(n-1)/2$ internet links.

The GPU solvers by themselves are only capable of solving linear programs and are not capable of solving mixed integer problems. Unfortunately, due to this fact shipping links are not included. Shipping links are what cause the problem to become a mixed integer program. A binary variable is used to mark if the shipping link will be used or not (e.g., 0 would denote the link is not used and 1 would denote the link is used). Branch and Bounds [39] (B&B) is a typical algorithm used to solved mixed integer programs. It chooses a set of $1s$ and $0s$ for the binary variables and solves a linear program and then compares the answer with another answer from a linear program with a different set of $1s$ and $0s$ for the binary variables. This process repeats while it prunes the set of $1s$ and $0s$, until an optimal solution is found. Since solving a mixed integer program with branch and bounds repeatedly solves independent linear programs, a decrease in computation time for solving a linear program translates into a decrease in computation time for a mixed integer program. So even though the GPU solvers cannot solve mixed integer programs, they can solve one of the core steps

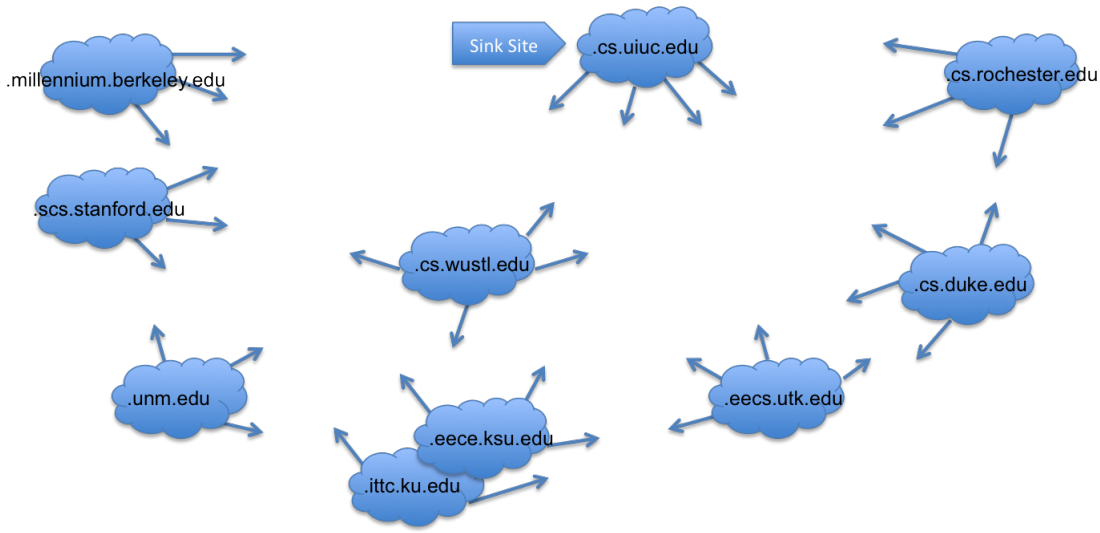


Figure 4.1: The Shipping Network

needed to solve a mixed integer program.

4.3 GPU Cluster

We evaluate the different GPU linear program solvers on the National Center for Supercomputing Application's (NCSA's) Intel 64 Tesla Linux Cluster Lincoln [37, 19] because of its computing power and computational resources. The machines at NCSA's Lincoln cluster are equipped with Intel 64 (Harpertown) 2.33 GHz dual socket quad core central processing units (CPUs), two Nvidia Tesla S1070 Accelerator Units with 16GB memory, 16GB of main memory, and run Red Hat Enterprise Linux 4 (Linux 2.6.19). This hardware was chosen because the size of the problem's matrix gets very large and increases as the time deadline and shipping network increases. The hardware needs to be capable of holding the problem's matrix in memory and performing operations on it and this cluster was the best resource that could be obtained. Furthermore, NCSA's Lincoln cluster is a TeraGrid [6] resource which is a project that aims to provide high performance computing resources for scientific computing research. Finally, in order to compare the performance of GPU linear programming solvers with a CPU implementation of a linear programing solver, we evaluated the computation time of GLPK on an Intel Core i5 520M (2.4GHz) with 8GB of main memory

running Mac OS X 10.6.

4.4 GNU Linear Programming Kit

As a baseline to compare the GPU solvers against, the GNU Linear Programming Kit (GLPK) is used to evaluate the computation time of a linear program solver on a CPU.

4.4.1 Problem Size

It is important to understand the size of the problem under certain constraints of a time expanded shipping network because the size of the problem determines the amount of memory needed to store the problem and corresponds to an increase in computation time needed to solve the problem. The size of problem is stated as the number of rows and columns in the A matrix, which is in turn determined by the number of links and sites used in the time expanded shipping network. The number of rows and columns used in the GLPK problem's matrix is shown in Figure 4.2. Since GLPK supports row equality constraints, the number of rows in A is equivalent to the number of sites plus the number of links in a time expanded shipping network. The number of columns is equal to the number of links in a time expanded shipping network. Furthermore, the number of sites and links in a time expanded shipping network is determined by the total time deadline, as their is a copy of each site at each time point in the total time deadline. Since both the rows and the columns grow linearly as the total time deadline increases linearly, the problem size grows exponentially. Finally, the total demand (i.e., the total data to transfer) does not affect the size of the matrix because the demand is just placed in parts of the b vector.

GLPK uses a sparse matrix format to store the problem, so despite the large problem size, the memory usage is proportional to the number of non zero entries in A.

4.4.2 Computation Time

The computation times for GLPK with various size shipping networks, time deadlines, and data demands, is shown in Figure 4.3. The charts show that the computation times for

GLPK linearly increases as a function of the problem size. The amount of data to transfer has little to no affect on the computation time. Overall the computation time for GLPK is fairly low with a maximum computation time of about 5 seconds for a 40000×25000 matrix made from a time expanded shipping network of 9 sites and a deadline of 216 hours (see Figure 4.3h).

4.5 GPU Simplex

The GPU Simplex [3] solver uses a revised simplex method implemented on a GPU using shaders for computation.

4.5.1 Problem Size

The number of rows and columns used in the GPU Simplex problem's matrix is shown in Figure 4.4. Unfortunately, the problem size for GPU Simplex contains more rows than the problem size for GLPK because the GPU Simplex method does not support row equality constraints. In order to add row equality constraints needed for conservation rows, both an upper bound and a lower bound are added for each conservation row. This makes the number of rows equal to two times the number of sites in a time expanded shipping network plus the number of links in a time expanded shipping network.

The GPU Simplex solver does not use a sparse matrix format to store the problem. Thus, for large problem sizes a large amount of main memory and GPU memory are required. This makes the GPU Simplex solver challenging to use since it is very computationally resource demanding. At the time of writing, the only resource available that could meet these demands was the GPU cluster described earlier. Even then, the GPU cluster is not immediately available due to user-resource management software (e.g., 1 to 2 hour wait time for a 30 minute wall time).

4.5.2 Computation Time

The computation times for large input sizes are shown in Table 4.1. Some data points in

# Sources	Time Deadline (hours)	Average Computation Time (secs)
1	120	15.07
2	24	1.97
2	48	5.77
2	72	11.98
3	24	3.30
3	48	9.94
4	24	9.74
5	24	7.11
6	24	9.70
7	48	59.89

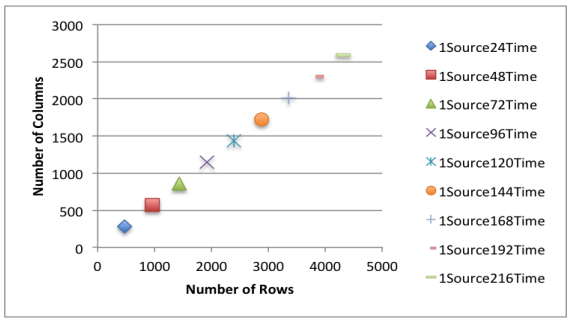
Table 4.1: GPU Simplex's average computation time for a given number of sources and time deadline (hours)

the table are missing because of scalability issues with the GPU Simplex solver. When the GPU Simplex solver ran on the same input as the GLPK solver, the computation time ranged between 2 seconds and 60 seconds. Since so many data points were missing, it was hard to evaluate the solver. Smaller problem sizes worked properly on the GPU Simplex solver, so in order to proceed with evaluations, the shipping network was adjusted to be expanded over a smaller total time and less data was sent from each source. The computation times and average computation times in seconds for smaller shipping networks are shown in Figure 4.5 and Figure 4.6, respectively. The computation time from smaller shipping networks increases linearly as a function of the problem size. However, the computation times for the GPU Simplex solver are higher than the computation times for GLPK. We speculate that this happened because of an incurred overhead for transferring a large matrix from main memory to the GPU's memory. The evaluation of the GPU Simplex solver shows that it does not scale to shipping networks with a time deadline greater than 72 hours.

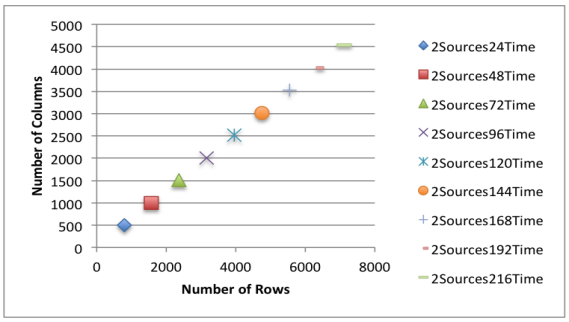
4.6 CUDA Linear Program Solver

This thesis attempted to evaluate the performance of the CUDA Linear Program Solver [47]. However, significant roadblocks were encountered during the process of integration that prevented it from working on the bulk data transfer problem. When the solver was given

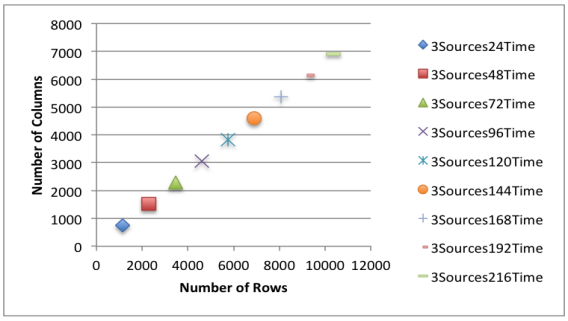
negative numbers in the A matrix, the solver would incorrectly place negative values in the x vector. In order for the CUDA solver to solve the bulk data transfer problem, it will need to be changed such that it works with negative numbers. We believe that changing the CUDA Linear Program Solver to work on Pandora's workloads is a direction for significant future work.



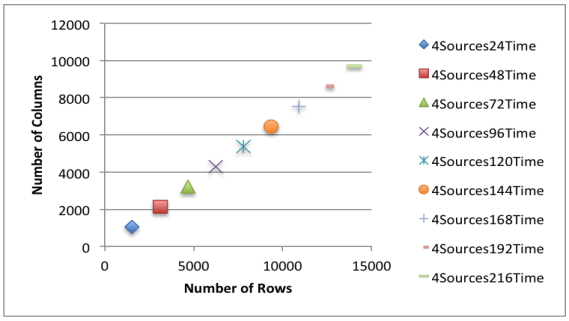
(a) 1 Source



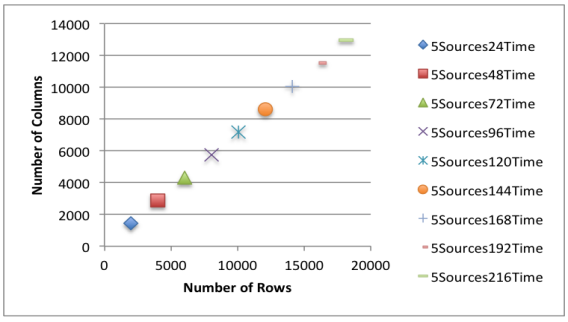
(b) 2 Sources



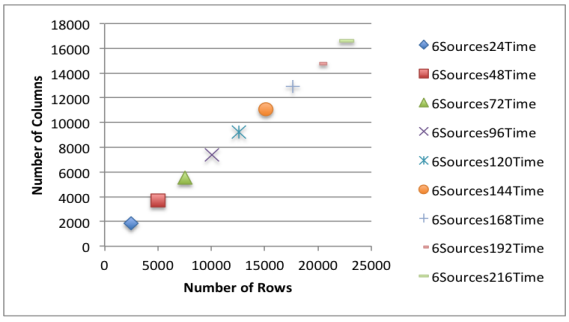
(c) 3 Sources



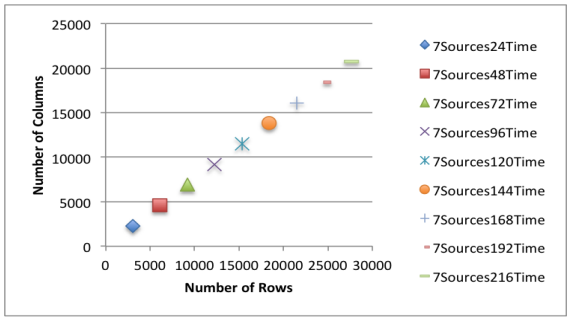
(d) 4 Sources



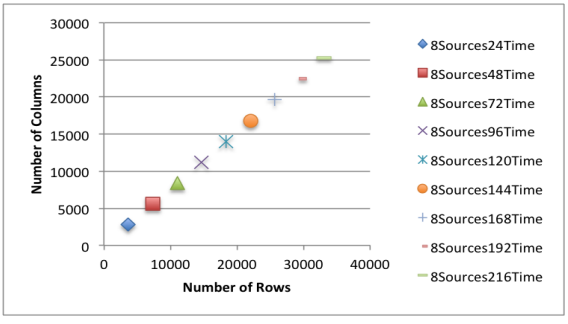
(e) 5 Sources



(f) 6 Sources

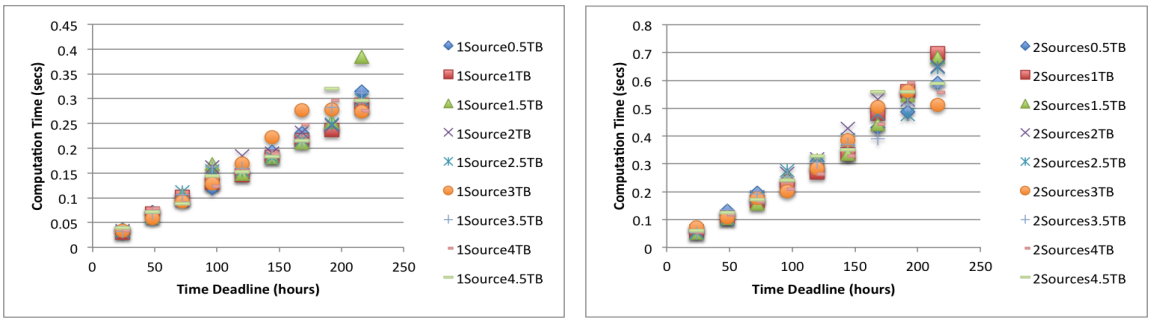


(g) 7 Sources



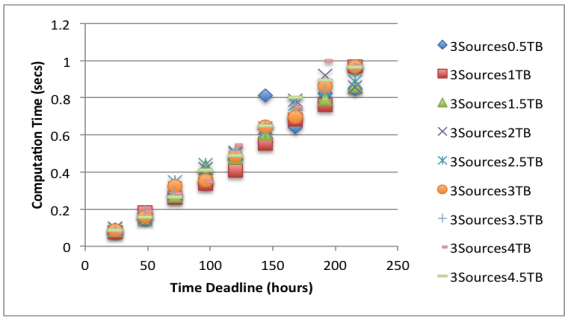
(h) 8 Sources

Figure 4.2: GLPK's problem size (i.e., number of rows and columns) based on the number of sources and total time

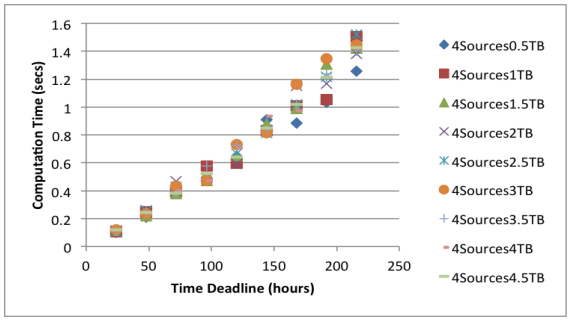


(a) 1 Source

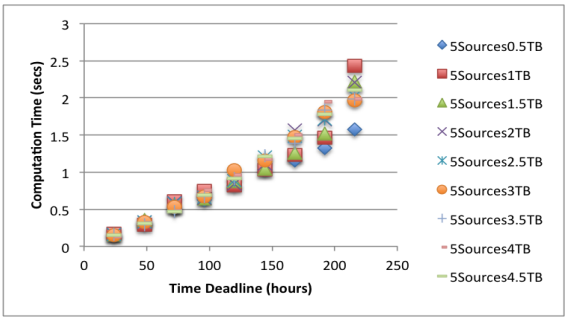
(b) 2 Sources



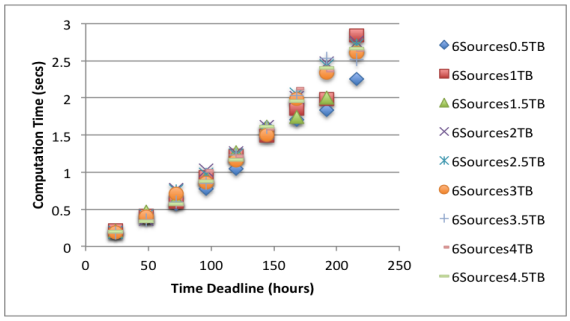
(c) 3 Sources



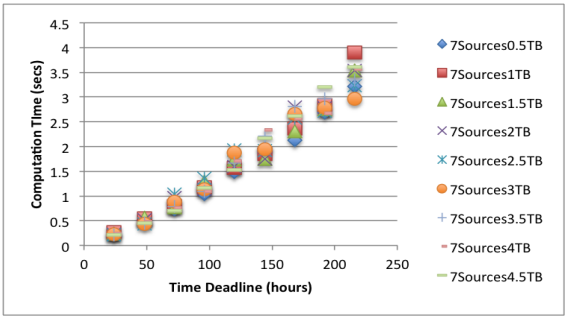
(d) 4 Sources



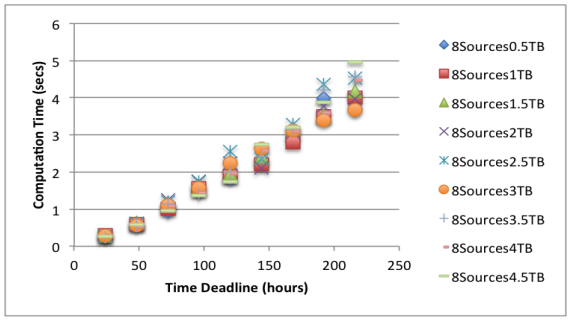
(e) 5 Sources



(f) 6 Sources

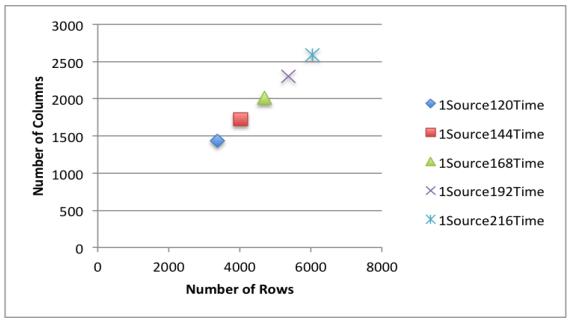


(g) 7 Sources

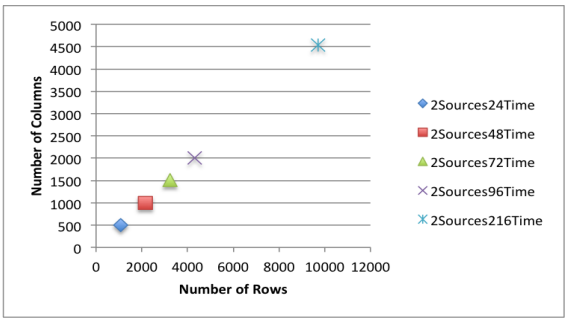


(h) 8 Sources

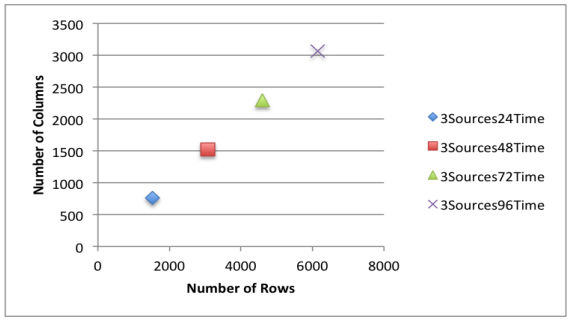
Figure 4.3: GLPK's computation time based on number of sources, total time, and data transfered



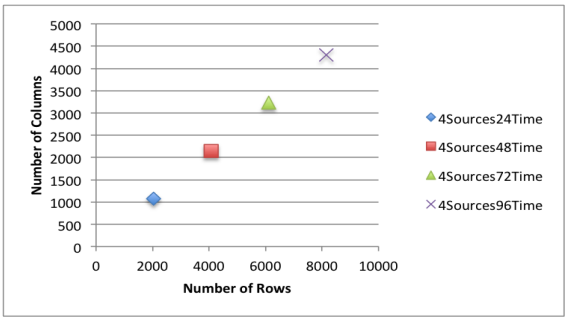
(a) 1 Source



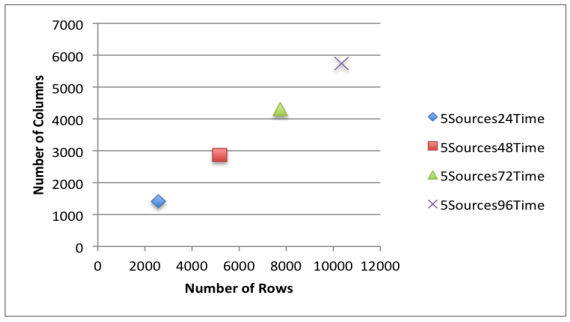
(b) 2 Sources



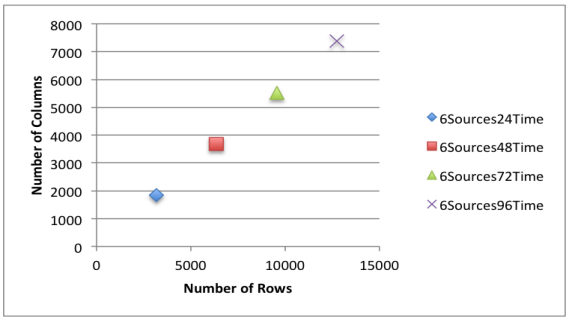
(c) 3 Sources



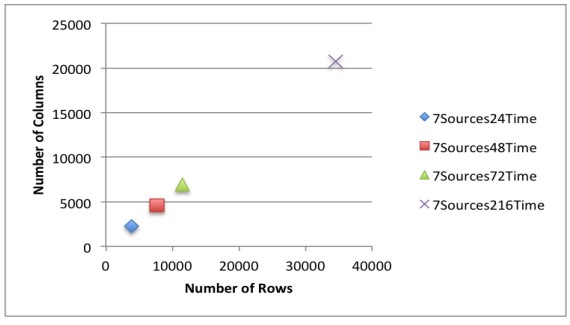
(d) 4 Sources



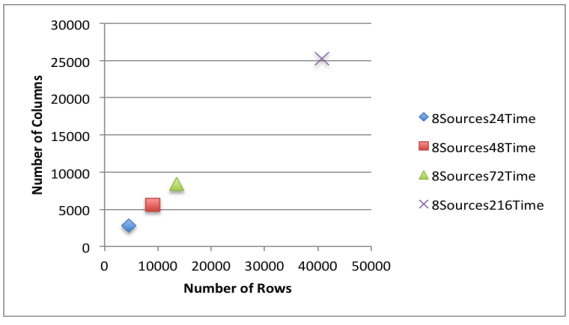
(e) 5 Sources



(f) 6 Sources

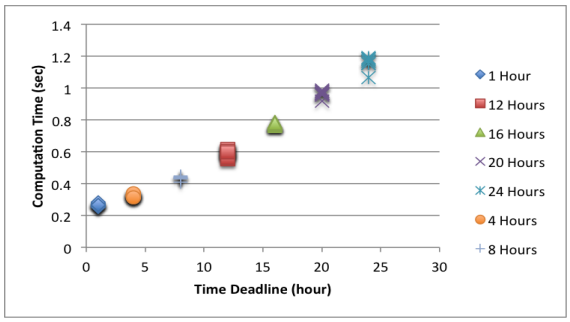


(g) 7 Sources

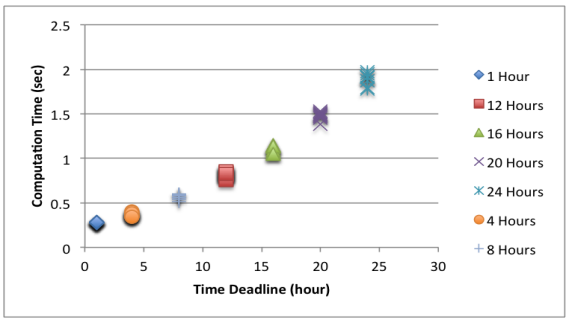


(h) 8 Sources

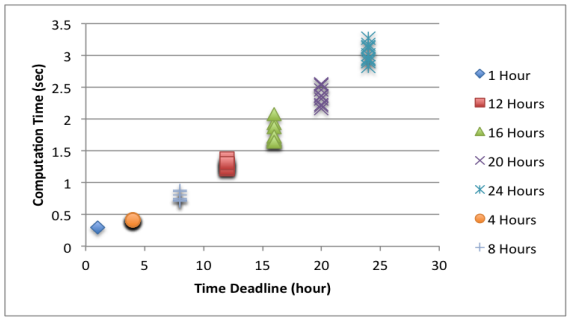
Figure 4.4: GPU Simplex's problem size (i.e., number of rows and columns) based on the number of sources and total time



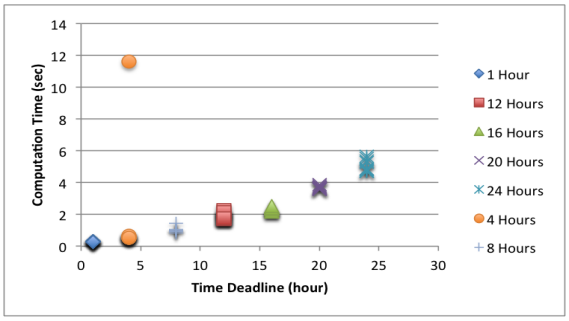
(a) 1 Source



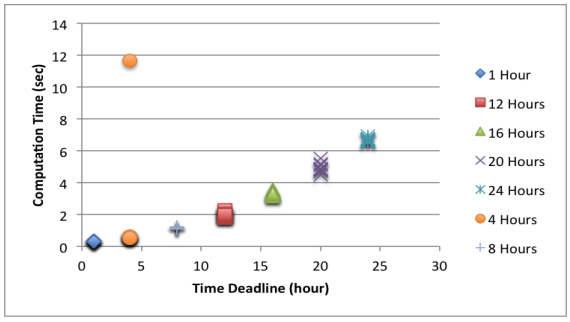
(b) 2 Sources



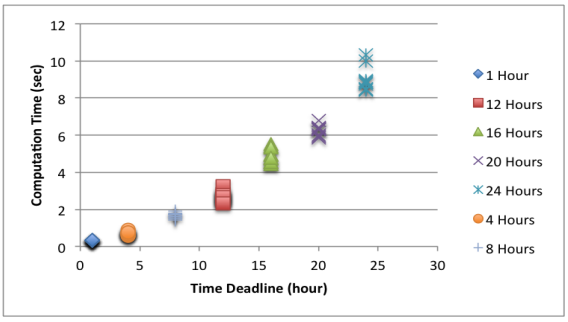
(c) 3 Sources



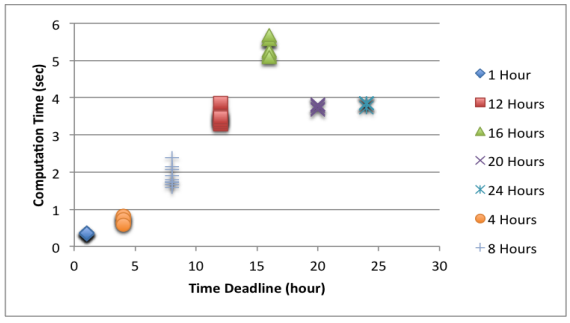
(d) 4 Sources



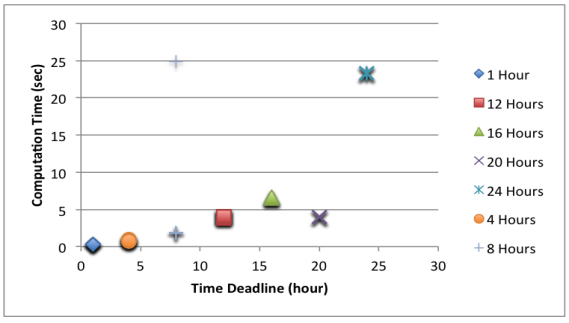
(e) 5 Sources



(f) 6 Sources



(g) 7 Sources



(h) 8 Sources

Figure 4.5: GPU Simplex's computation time (secs) based on varying number of sources and smaller time deadline (hours)

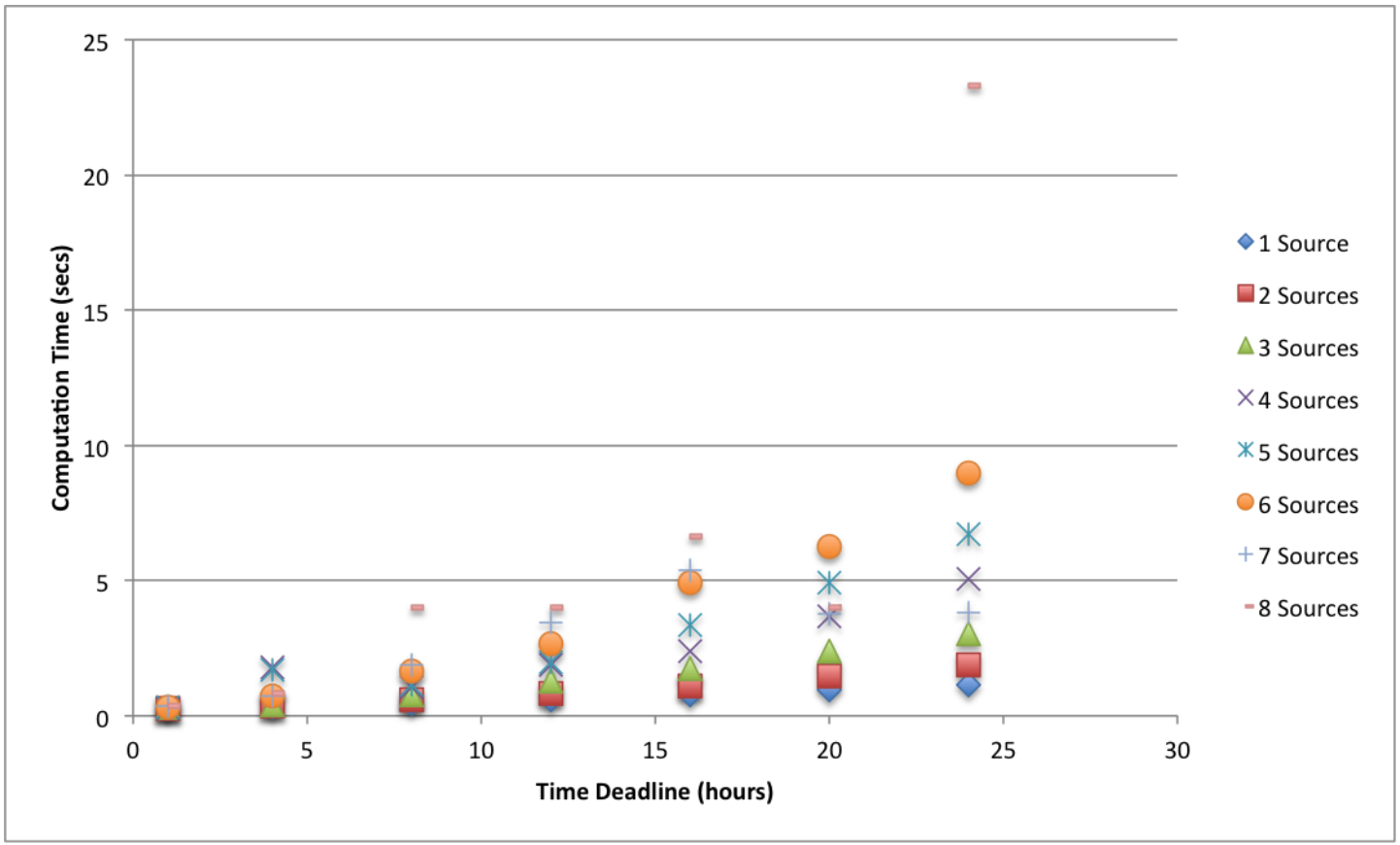


Figure 4.6: GPU Simplex's average computation time (secs) based on varying number of sources and small time deadline (hours)

Chapter 5: Related Work

This thesis builds on top of and extends Pandora [9], which is the first system to propose, model, and solve the bulk data transfer problem. Similar to Pandora, this thesis solves the same bulk data transfer planning problem. However, this thesis is more focused on providing a better interface for the problem, and evaluating GPU solvers on its workloads.

Many technologies are used to build the web service described in Chapter 2. Common approaches for web services that operate over HTTP [17] use a REST [18] style architecture, which serves human readable data formats such as XML [4], Atom [43], RSS [48], and JSON [12]. The core of the web service was written using a web framework called Django [23] that decreases development time, and was served from an Apache HTTP [22] server using mod_wsgi [14]. All of the data was persistently stored in a MySQL [11] database. However, web services that need to scale to a large number of users can substitute MySQL for a distributed data store system (e.g., Google's BigTable [7] via Google AppEngine [32, 33], Apache's CouchDB [20, 8], Apache's Cassandra [38, 46]). In order to visualize and provide a human usable interface into the data model, a web service typically provides a web site that is made with XHTML [1], CSS [40], and JavaScript [35]. To make the web site more dynamic and aesthetically appealing, jQuery and jQuery UI [44] were used. Alternatively, the web service can have a human usable interface that is tailor for mobile devices via Google's Android [31] or via Apple's iOS [27]. Finally, web services need a way to authenticate their users, rather than building new authentication systems that need to store personal user information, it can be more secure and reliable to use other more established authentication

services. In particular, the web service in Chapter 2 uses Facebook's authentication [28], which uses the OAuth [15] protocol.

The web service described in Chapter 2 provides an effective interface to determine the optimal way to transfer large amounts of data. The web service itself is not concerned with actually transferring any of the data (either through hard drive shipments or the internet), but does give users of the service an optimal plan or strategy to perform the transfer. Other related services like Globus Online [30], FedEx's [10], and UPS's [34] web service, concern themselves with the process of reliably and effectively transferring the data. The web service described in this thesis relies on and could collaborate with the related services to benefit users interested in transfers large amounts of data over a wide area network.

Globus Online [30] is a service that provides high performance, reliable, and secure data movement. Globus Online focuses on researchers who need to move large amounts of their research data to various geographically distant sites. Users of Globus Online can pick the they files want to transfer and where they want them to be transfered to and from, and Globus Online will take care of the rest. It is reliable, secure, fast, and easy to use. In order to use the service, users set up endpoints at their sites that need data to be transfered to and from and the service reliability transfers the data between them. Similar to Pandora, users can interact with the system through either a web interface or a command line interface. However, unlike Pandora, Globus Online does not support an API, which forces users to only be able to interact with their services in certain ways, and does not provide an easy way to collaborate with other services. Unlike Pandora, Globus Online solely focuses on the physical act of transferring data between two points across the internet. Pandora focuses on constructing a transfer plan that details how data should be transfered across various end points under a certain time deadline or budget constraint. There would be great benefit in using Globus Online and Pandora in conjunction. Specifically, a user could use Pandora to obtain a transfer plan that details the times and locations of where to transfer data to and from, and then use Globus Online to perform the internet transfers outlined in the transfer

plan. Alternatively, without Globus Online users of Pandora have to set up their own means of transferring data (e.g., scp, ftp) and make sure it reliably transfers.

Both FedEx [10] and UPS [34] provide services that allow users of their system to schedule shipments and receive shipment prices. Similar to Globus Online, FedEx and UPS are only concerned with physically shipping packages from one location to another under a certain time deadline based on a service level (e.g., ground, overnight, two day). Pandora formulates the shipping plan and FedEx or UPS performs the actual act of shipping.

This thesis evaluates the performance of new linear program solvers on the bulk data transfer problem. The original Pandora system used a CPU based implementation [29] and this thesis evaluates the performance of newer GPU linear program solvers that use the simplex method [3, 47]. There is another GPU solver [26] that uses the interior point method that was not evaluated because the program or its source code could not be obtained. GPU based solvers were investigated because they provide high amounts of parallelism and they have become increasingly popular for many computing tasks. In particular, the evaluation of these solvers showed promising results over CPU implementations [29, 45]. The main goal was to lower the computation time of solving the bulk data transfer problem.

In order to perform the evaluations, a large amount of computational resources were needed. We used NCSA's Lincoln GPU cluster [37, 19] because of the vast amount of computational resources it provides. NCSA's Lincoln GPU cluster is part of the TeraGrid [6] project that is focused on providing computational resources to researchers.

The main reason this thesis focused on GPUs was because of the high amounts of parallelism they provide. A distributed system of machines is another opportunity to leverage high amounts of parallelism. There is recent work that implements branch and bound [39] algorithms on MapReduce [13] frameworks [42, 5] (e.g., Hadoop [21] and Dryad [36]). This work differs from the GPU solvers because they are not focused on solving linear programs. Instead their focus is on branch and bounds, which is an algorithm used in solving mixed integer programs. Evaluations of this work combined with either the GPU or CPU linear

program solver on the bulk data transfer problem would present interesting results.

Chapter 6: Conclusion

This thesis explored the design and implementation of two systems, and evaluated the performance of various GPU linear program solvers. The first system was a web service that allows users to plan bulk data transfers through a web interface and a RESTful interface. The second system was Pandora's Toolbox, a modular and extensible C++ framework that models the bulk data transfer problem and allows various types of linear programming solvers to easily integrate into it. The evaluation of the various GPU linear program solvers reveals reasonable performance for small shipping networks and small time deadlines, however with scalability limitations.

There are many more enhancements that could be added to Pandora's web service and to the GPU solvers that is left as future work. Pandora's web service could visualize the shipping network and shipping plan by using graph visualization libraries [16, 25, 41]. The web service could implement or use a system that in real time automatically calculates and updates the bandwidth information for sites and the links between them. The web service could return other formats (e.g., XML [4], Atom [43], and RSS [48]), which would allow users to choose their preferred data format. A new service for the web service could be implemented or made via existing technologies (e.g., Globus Online [30], FedEx [10], and UPS [34]) to perform data transfers and shipments on behalf of users. The web service could provide other user authentication systems that allow alternative ways for users to authenticate. In order to solve mixed integer programs with the GPU solvers, one could implementation a branch and bounds algorithm [39] that uses the GPU solvers, or use the

Symphony library [45] with the GPU solvers as a black box LP solver. There has been work in implementing branch and bounds algorithms on distributed execution engines [5, 42] (e.g., MapReduce [13] paradigm frameworks like Hadoop [21] and Dryad [36]) that could reduce the computation time of the bulk data transfer problem with shipping links. Finally, this thesis leaves the issue of adapting the various GPU linear program solvers [3, 47] to scale to the bulk data transfer problem's workloads as future work.

Bibliography

[1] M. Baker and P. Stark. The 'application/xhtml+xml' Media Type. RFC 3236 (Informational), January 2002.

[2] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). RFC 1738 (Proposed Standard), December 1994. Obsoleted by RFCs 4248, 4266, updated by RFCs 1808, 2368, 2396, 3986, 6196.

[3] J. Bieling, P. Peschlow, and P. Martini. An efficient gpu implementation of the revised simplex method. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1 –8, April 2010.

[4] J. Boyer. Canonical XML Version 1.0. RFC 3076 (Informational), March 2001.

[5] Mihai Budiu, Daniel Delling, and Renato Werneck. DryadOpt: Branch-and-bound on distributed data-parallel execution engines. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Anchorage, AK, May 16-20 2011.

[6] Charlie Catlett. The philosophy of teragrid: Building an open, extensible, distributed terascale facility. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '02, pages 8–, Washington, DC, USA, 2002. IEEE Computer Society.

[7] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[8] Benot Chesneau. Couchdbkit. `http://couchdbkit.org/`. a framework to allow your python application to use Apache CouchDB.

[9] Brian Cho and Indranil Gupta. New algorithms for planning bulk transfer via internet and shipping networks. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pages 305 –314, June 2010.

[10] FedEx Corporation. Fedex web services for shipping. http://fedex.com/developer/. Interface to FedEx-hosted functionality used by developers to integrate FedEx shipping or FedEx Office document printing and finishing functionality into software programs, web sites or web applications.

[11] Oracle Corporation. Mysql database. http://www.mysql.com/. the world's most popular open source database software.

[12] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006.

[13] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.

[14] Graham Dumpleton. mod wsgi. http://www.modwsgi.org/. a simple to use Apache module which can host any Python application which supports the Python WSGI interface.

[15] Ed. E. Hammer-Lahav, Yahoo!, D. Recordon, Facebook, D. Hardt, and Microsoft. The oauth 2.0 authorization protocol draft-ietf-oauth-v2-15. http://tools.ietf.org/html/draft-ietf-oauth-v2-15, 2011.

[16] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen North, Gordon Woodhull, Short Description, and Lucent Technologies. Graphviz open source graph drawing tools. In *Lecture Notes in Computer Science*, pages 483–484. Springer-Verlag, 2001.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785.

[18] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

[19] The National Center for Supercomputing Applications. Intel 64 tesla linux cluster. http://www.ncsa.illinois.edu/UserInfo/Resources/Hardware/Intel64TeslaCluster/.

[20] Apache Software Foundation. Apache couchdb. http://couchdb.apache.org/. a peer-based distributed database system.

[21] Apache Software Foundation. Apache hadoop. http://hadoop.apache.org/. The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing.

[22] Apache Software Foundation. Apache http service project. http://httpd.apache.org/. an open-source HTTP server for modern operating systems including UNIX and Windows NT.

[23] Django Software Foundation. Django. http://www.djangoproject.com/. a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

[24] Pascal-Emmanuel Gobry. Gawker hacked! http://www.businessinsider.com/gawker-hacked-2010-12.

[25] Jeffrey Heer, Stuart K. Card, and James Landay. Prefuse: A toolkit for interactive information visualization. In *ACM Human Factors in Computing Systems (CHI)*, pages 421–430, 2005.

[26] Jin Hyuk, Jung, Dianne P. Oleary, Dedicated Gene, and H. Golub. Implementing an interior point method for linear programs on a cpu-gpu system .

[27] Apple Computer Inc. ios. http://www.apple.com/ios/. An operating system for mobile devices.

[28] Facebook Inc. Facebook authentication. https://developers.facebook.com/docs/authentication/, 2011.

[29] Free Software Foundation Inc. Gnu linear programming kit. http://www.gnu.org/software/glpk/. intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems.

[30] Globus Consortium Inc. Globus online. http://www.globusonline.org/.

[31] Google Inc. Android. http://www.android.com/. An operating system for mobile devices.

[32] Google Inc. Google app engine. https://appengine.google.com/. build and host web apps on the same systems that power Google applications.

[33] Google Inc. Google app engine documentation: How entities are stored. http://code.google.com/intl/pl/appengine/articles/storage_breakdown.html#anc-background. App Engine's datastore is built on top of Bigtable, a large, distributed key-value store that is built to scale.

[34] United Parcel Service Inc. Ups developer kit. https://www.ups.com/upsdeveloperkit. programming instructions and standards for accessing and integrating UPS functionality into e-commerce websites or enterprise applications.

[35] ECMA International. *Standard ECMA-262*. ECMA International, 1999.

[36] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. *SIGOPS Oper. Syst. Rev.*, 41:59–72, March 2007.

[37] V.V. Kindratenko, J.J. Enos, Guochun Shi, M.T. Showerman, G.W. Arnold, J.E. Stone, J.C. Phillips, and Wen mei Hwu. Gpu clusters for high-performance computing. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pages 1 –8, 31 2009-sept. 4 2009.

[38] Avinash Lakshman and Prashant Malik. Cassandra: structured storage system on a p2p network. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, PODC '09, pages 5–5, New York, NY, USA, 2009. ACM.

[39] A. H. Land and A. G Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[40] H. Lie, B. Bos, and C. Lilley. The text/css Media Type. RFC 2318 (Informational), March 1998.

[41] Tamara Munzner. Drawing large graphs with h3viewer and site manager (system demonstration), 1998.

[42] Derek G. Murray and Steven Hand. Non-deterministic parallelism considered useful. In *HotOS XIII, 13th Workshop on Hot Topics in Operating Systems*, 2011.

[43] M. Nottingham and R. Sayre. The Atom Syndication Format. RFC 4287 (Proposed Standard), December 2005. Updated by RFC 5988.

[44] The JQuery Project and The JQuery UI Team. Jquery ui. http://jqueryui.com/. abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets, built on top of the jQuery JavaScript Library, that you can use to build highly interactive web applications.

[45] T K Ralphs and M Güzelsoy. The SYMPHONY callable library for mixed-integer linear programming. In *Proceedings of the Ninth INFORMS Computing Society Conference*, pages 61–76, 2005.

[46] rob.vaterlaus@gmail.com. Django cassandra backend. https://github.com/vaterlaus/django_cassandra_backend. A Django database backend for Cassandra.

[47] Daniele G. Spampinato and Anne C. Elstery. Linear optimization on modern gpus. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[48] Harvard University. Really simple syndication 2.0. http://cyber.law.harvard.edu/rss/rss.html.

[49] Michael Vrable, Stefan Savage, and Geoffrey M. Voelker. Cumulus: Filesystem backup to the cloud. *Trans. Storage*, 5:14:1–14:28, December 2009.